	Hw 1	Sorash Khadem

1.) a.) (see attached picture)

b.) Horner's method is exactly the same as numpy's polyval, probably b/c numpy uses Horner's rule. Evaluating $p(x)$ directly as $(x-2)^9$ seems the most correct, since there are no jumps. Since 2 is a root, the other methods most likely suffer from rounding errors.

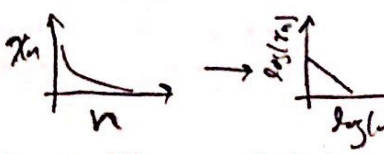
2.) a.) $f(x) = \sqrt{x+1} - 1 = \dfrac{(\sqrt{x+1}-1)(\sqrt{x+1}+1)}{(\sqrt{x+1}+1)} = \dfrac{x}{\sqrt{x+1}+1}$

↳ This avoids cancellation, since $x \approx 0$ does not have any subtraction by very close numbers

b.) $f(x) = \sin(2(x+a)) - \sin(2a) = 2\sin\left(\frac{2x+4a}{2}\right)\cos\left(\frac{2x}{2}\right)$

$= 2\sin(x+2a)\cos(x)$

↳ Now, as $x \to 0$, $\cos(x)$ is well-defined. There is no longer a difference, so no cancellation.

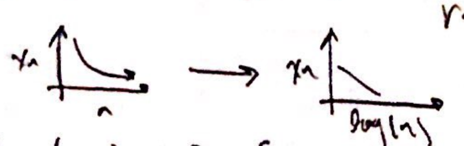
3.) a.) $x_n = Cn^{-\alpha}$ 

$\Rightarrow$ Both

Just y: $\log(x_n) = Cn^{-\alpha}$ => right side grows much faster

Just x: $x_n = -\alpha \log(Cn)$ => left side grows faster

Both: $\log(x_n) = -\alpha \log(Cn)$ => Both grow $O(\log)$, so same rate.

b.) $x_n = Dp^n$ 

=> x axis

↳ $x_n = n \log(Dp)$ => grow at same rate.

c.) This series is linearly convergent (see jupyter nb) since it is linear on a log-lin plot. This fits (2), with $D = 5.6$, and $p = \frac{4.48}{5.6} = 0.8$

d.) This Series is sublinear since it is a straight line on a log-log plot. Therefore, it is of the form (1), with $C = 3$ (if n starts at 1):

$$1.0607 = 3(2)^{-\alpha} \Rightarrow -\alpha = \frac{\log(1.0607/3)}{\log(2)} \approx 1.5$$

4.) $$\frac{1}{1-h} - h - 1 = \frac{1}{1-h} - \frac{h-h^2}{1-h} - \frac{1-h}{1-h} = \frac{h^2}{1-h} = O(h)$$

Check: $\lim_{h\to 0} \frac{h^2}{1-h} \cdot \left(\frac{1}{h}\right) = \lim_{h\to 0} \frac{h}{1-h} = \lim_{h\to 0} \frac{1}{1/h - 1} = 0 < \infty$ ✓

5.) $f(x) = e^x - 1$

a.) $K_f(x) = \left|\frac{x}{f(x)} \cdot f'(x)\right| = \left|\frac{x}{e^x - 1} \cdot e^x\right|$

This expression is ill-conditioned for $x \approx 0$

b.) Alg: ① $y = e^x \rightarrow K_g(x) = |x|$

② $g = y - 1 \rightarrow K_g(x) = \left|\frac{x}{x-1}\right|$

} This algorithm is unstable, since K_g is more than K_f for $x \approx 1$

c.) $x = 9.999999995 \times 10^{-10} \rightarrow f(x) = 10^{-9}$

Using $e^x - 1$, a Float64 gives 7.082 correct digits.

Plugging into condition number: $K_f(x) \approx$

d.) $f(x) \approx x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24} \ldots$ (T.S. Centered at $x=0$ for $e^x - 1$)

$$|R_n| \le \frac{1}{(n+1)!} \underbrace{e^{\xi}}_{\text{Bounded by } 2} x^{n+2} \le \frac{2}{(n+1)!} \underbrace{x}_{\text{bounded by } 10^{-9}}^{n+1}$$

$|R_n| \le \frac{2}{(n+1)!}(10^{-9})^{n+1} \rightarrow$ want this to be $\le 10^{-16}$

Checking $n=1$, we see that $|R_n| \le$ 1e-18, so two terms is enough:

$$f(x) = x + \frac{x^2}{2}$$ (Note: See attached Jupyter Notebook)

e.) Plugging in the x value from before, we get 18 digits of precision.

APPM 4650 HW 1

Soroush Khadem

Note that the handwritten pages are supplemental

In [44]:
```
import numpy as np
import math
import matplotlib.pyplot as plt

plt.rcParams['figure.figsize'] = [15, 5]
```

Problem 1

In [45]:
```
# Full written out form
def naive(x):
    return math.pow(x, 9) - 18*math.pow(x, 8) + 144*math.pow(x, 7) - 672*math.pow(x, 6) + 2016*math.po
w(x, 5) - 4032*math.pow(x, 4) + 5376*math.pow(x, 3) - 4608*math.pow(x, 2) + 2304*x - 512
```

In [46]:
```
# Compact form
def p(x):
    return math.pow(x-2, 9)
```

In [47]:
```
# Horner's method
def horners(x):
    rv = np.poly([2]*9)[0]
    for i in range(1, 10):
        rv = rv*x + np.poly([2]*9)[i]
    return rv
```

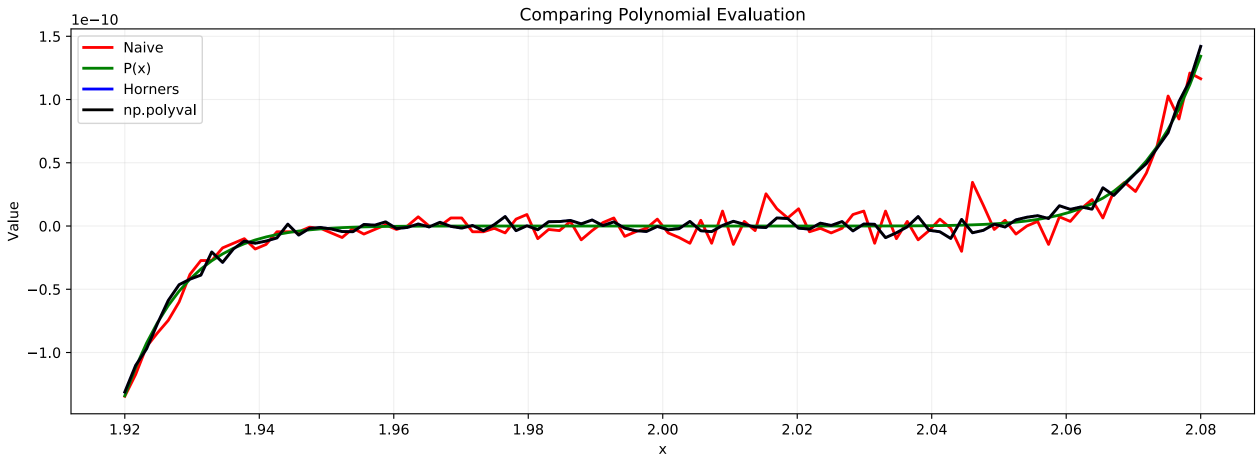
In [48]:
```
# Built-in polyval
def polyval(x):
    return np.polyval(np.poly([2]*9), x)
```

In [49]:
```
# Evaluate near the roots
x = np.linspace(1.92, 2.08, 100)
naive_y = [naive(i) for i in x]
p_y = [p(i) for i in x]
horners_y = [horners(i) for i in x]
polyval_y = [polyval(i) for i in x]
```

In [50]:
```
# Plot results
fig = plt.figure()
plt.plot(x, naive_y, 'r', linewidth=2)
plt.plot(x, p_y, 'g', linewidth=2)
plt.plot(x, horners_y, 'b', linewidth=2)
plt.plot(x, polyval_y, 'k', linewidth=2)
legend_text = ['Naive', 'P(x)', 'Horners', 'np.polyval']

plt.grid(True, alpha=0.2)
plt.title('Comparing Polynomial Evaluation')
plt.xlabel('x')
plt.ylabel('Value')
plt.legend(legend_text)
```

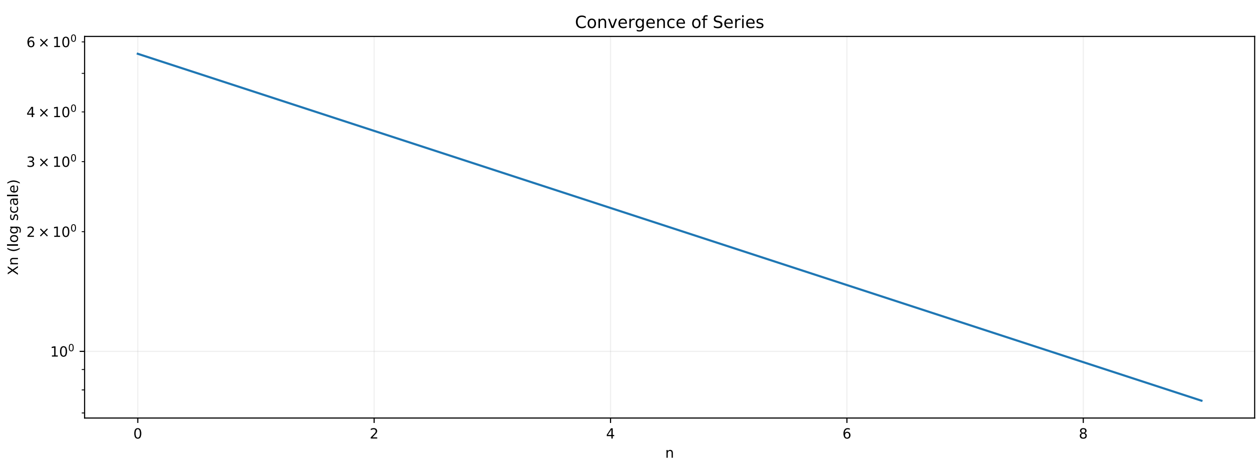
Out[50]: <matplotlib.legend.Legend at 0x209de0c61d0>



Problem 3

In [51]:
```
xn = [5.6, 4.48, 3.584, 2.8672, 2.2938, 1.8350, 1.4680, 1.1744, 0.9395, 0.7516]
plt.plot(xn)
plt.yscale('log')
plt.grid(True, alpha=0.2)
plt.title('Convergence of Series')
plt.xlabel('n')
plt.ylabel('Xn (log scale)')
```

Out[51]: Text(0,0.5,'Xn (log scale)')

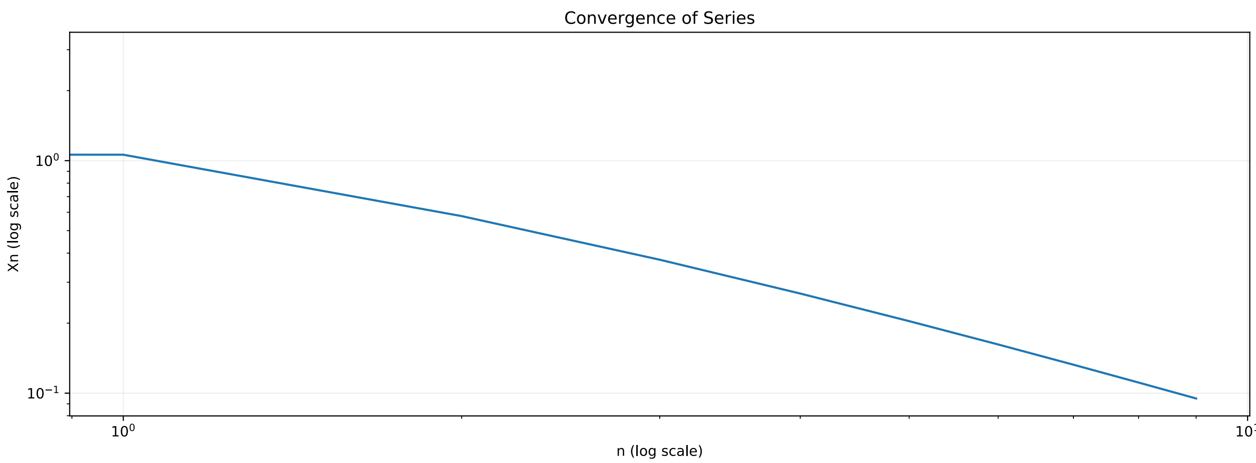


In [52]:
```
xn[1] / xn[0]
```

Out[52]: 0.8000000000000002

In [53]:
```python
xn = [3, 1.0607, 0.5774, 0.3750, 0.2683, 0.2041, 0.1620, 0.1326, 0.1111, 0.0949]
plt.plot(xn)
plt.xscale('log')
plt.yscale('log')
plt.grid(True, alpha=0.2)
plt.title('Convergence of Series')
plt.xlabel('n (log scale)')
plt.ylabel('Xn (log scale)')
```

Out[53]: `Text(0,0.5,'Xn (log scale)')`



In [54]:
```python
-math.log(xn[1]/xn[0])/math.log(2)
```

Out[54]: `1.4999458272324424`

Problem 5

In [55]:
```python
x_val = np.float64(9.999999995e-10)
trueAnswer = 1e-9
relAccuracy = lambda x : np.abs(x-trueAnswer)/np.abs(trueAnswer)
numDigits   = lambda x : -np.log10( relAccuracy(x) + 1e-18 )
f = lambda x : np.exp(x) - 1.0
numDigits(f(x_val))
```

Out[55]: `7.082282536427183`

In [56]:
```python
condition = lambda x : np.abs((x/trueAnswer)*np.exp(x))
condition(x_val)
```

Out[56]: `1.0000000005`

In [57]:
```python
# Bounding the error on the Taylor series: |Rn| < 2/(n+1)! * x^(n+1)
def find_n(n):
    return (2/math.factorial(n+1)) * (1e-9)**(n+1)
```

In [58]:
```python
find_n(1)
```

Out[58]: `1e-18`

In [59]:
```python
# Approximate using TS
def taylor(x):
    return x + (x**2 / 2)
```

In [60]:
```python
numDigits(taylor(x_val))
```

Out[60]: `18.0`